Date=6/11/2020
Lecture By=Manish Mahant
Subject ⇒ Class And Inheritance In Javascript

IN PREVIOUS LECTURE **(QUICK RECAP) Date-5/11/2020**	**In Today's Lecture (Overview)**

JavaScript Classes

ECMAScript 2015, also known as ES6, introduced JavaScript Classes.

JavaScript Classes are templates for JavaScript Objects.

JavaScript Class Syntax

Use the keyword `class` to create a class.

Always add a method named `constructor()`:

Syntax

```
class ClassName {
  constructor() { ... }
```

Example

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
}
```

The example above creates a class named "Car".

The class has two initial properties: "name" and "year".

Using a Class

When you have a class, you can use the class to create objects:

Example

```
let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);
```

The Constructor Method

The constructor method is a special method:

- It has to have the exact name "constructor"
- It is executed automatically when a new object is created
- It is used to initialize object properties

It if you do not define a constructor method, JavaScript will add an empty constructor method.

Class Methods

Class methods are created with the same syntax as object methods.

Use the keyword `class` to create a class.

Always add a `constructor()` method.

Then add any number of methods.

Syntax

```
class ClassName {
  constructor() { ... }
  method_1() { ... }
  method_2() { ... }
  method_3() { ... }
  }
}
```

Create a Class method named "age", that returns the Car age:

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    let date = new Date();
    return date.getFullYear() - this.year;
  }
}

let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
```

You can send parameters to Class methods:

Example

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}

let date = new Date();
let year = date.getFullYear();

let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML=
"My car is " + myCar.age(year) + " years old.";
```

Inheritance In Javascript

With most of the gory details of OOPS now explained, this article shows how to create "child" object classes (constructors) that inherit features from their "parent" classes. In addition, we present some advice on when and where you might use OOJS, and look at how classes are dealt with in modern ECMAScript syntax.

Prototypal inheritance

So far we have seen some inheritance in action — we have seen how prototype chains work, and how members are inherited going up a chain. But mostly this has involved built-in browser functions. How do we create an object in JavaScript that inherits from another object?

Let's explore how to do this with a concrete example.

Getting started

First of all, make yourself a local copy of our oojs-class-inheritance-start.html file (see it running live also). Inside here you'll find the same `Person()` constructor example that we've been using all the way through the module, with a slight difference — we've defined only the properties inside the constructor:

```
function Person(first, last, age, gender, interests) {
  this.name = {
    first,
    last
  };
  this.age = age;
  this.gender = gender;
  this.interests = interests;
};
```

The methods are *all* defined on the constructor's prototype. For example:

```
Person.prototype.greeting = function() {
  alert('Hi! I\'m ' + this.name.first + '.');
};
```

Defining a Teacher() constructor function

The first thing we need to do is create a `Teacher()` constructor — add the following below the existing code:

```
function Teacher(first, last, age, gender, interests, subject) {
  Person.call(this, first, last, age, gender, interests);

  this.subject = subject;
}
```

This looks similar to the Person constructor in many ways, but there is something strange here that we've not seen before — the `call()` function. This function basically allows you to call a function defined somewhere else, but in the current context. The first parameter specifies the value of `this` that you want to use when running the function, and the other parameters are those that should be passed to the function when it is invoked.

We want the `Teacher()` constructor to take the same parameters as the `Person()` constructor it is inheriting from, so we specify them all as parameters in the `call()` invocation.

The last line inside the constructor simply defines the new `subject` property that teachers are going to have, which generic people don't have.

As a note, we could have simply done this:

```
function Teacher(first, last, age, gender, interests, subject) {
  this.name = {
    first,
    last
  };
  this.age = age;
  this.gender = gender;
  this.interests = interests;
  this.subject = subject;
}
```

To Know More About Inheritance Please Visit The Link Below
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance

Property getters and setters

There are two kinds of object properties.

The first kind is *data properties*. We already know how to work with them. All properties that we've been using until now were data properties.

The second type of properties is something new. It's *accessor properties*. They are essentially functions that execute on getting and setting a value, but look like regular properties to an external code.

Getters and setters

Accessor properties are represented by "getter" and "setter" methods. In an object literal they are denoted by `get` and `set`:

```
let obj = {
  get propName() {
    // getter, the code executed on getting obj.propName
  },

  set propName(value) {
    // setter, the code executed on setting obj.propName = value
  }
}
```

The getter works when `obj.propName` is read, the setter – when it is assigned.

For instance, we have a `user` object with `name` and `surname`:

Now we want to add a `fullName` property, that should be `"John Smith"`. Of course, we don't want to copy-paste existing information, so we can implement it as an accessor:

```
let user = {
  name: "John",
  surname: "Smith",

  get fullName() {
    return `${this.name} ${this.surname}`;
  }
};

alert(user.fullName); // John Smith
```

From the outside, an accessor property looks like a regular one. That's the idea of accessor properties. We don't *call* `user.fullName` as a function, we *read* it normally: the getter runs behind the scenes.

As of now, `fullName` has only a getter. If we attempt to assign `user.fullName=`, there will be an error:

```
let user = {
  get fullName() {
    return `...`;
  }
};

user.fullName = "Test"; // Error (property has only a getter)
```

Let's fix it by adding a setter for `user.fullName`:

```
let user = {
  name: "John",
  surname: "Smith",

  get fullName() {
    return `${this.name} ${this.surname}`;
  },

  set fullName(value) {
    [this.name, this.surname] = value.split(" ");
  }
};

// set fullName is executed with the given value.
user.fullName = "Alice Cooper";

alert(user.name); // Alice
alert(user.surname); // Cooper
```

Accessor descriptors

Descriptors for accessor properties are different from those for data properties.

For accessor properties, there is no `value` or `writable`, but instead there are `get` and `set` functions.

That is, an accessor descriptor may have:

- **`get`** – a function without arguments, that works when a property is read,
- **`set`** – a function with one argument, that is called when the property is set,
- **`enumerable`** – same as for data properties,
- **`configurable`** – same as for data properties.

For instance, to create an accessor `fullName` with `defineProperty`, we can pass a descriptor with `get` and `set`:

To Know More About Getters and Setters Please Visit The Link Below
https://javascript.info/property-accessors

Questions For Self-Practice// CC And ASSIGNMENT For The Day

Assignment
https://au-assignment.s3.ap-south-1.amazonaws.com/Week_19_Day_5_Assignment-c0ed5486-bdeb-47ad-8644-b9f0d1fe3bc9.pdf
CC
https://au-assignment.s3.amazonaws.com/Week_19_Day_5_Challenge-ec611a17-c035-4b48-932a-4ecd8164c2b2.pdf